CS787: Advanced Algorithms	**Homework 5**
Out: 4/6/22	**Due: 4/14/22**

1 FPTAS for Knapsack

In the knapsack problem you are given n items each having weight w_i and value v_i and you are asked to compute the most valuable set with weight at most W. While this problem is NP-hard in general, you can compute a feasible subset having value at least $(1-\epsilon)$ times the optimal in time poly$(n, 1/\epsilon)$. You are asked to devise such an algorithm.

a. Let V be the value of the most valuable item with weight at most W. Show that rounding down the values of all items to the nearest multiple of $V\epsilon/n$ decreases the value of the optimum solution by at most a multiplicative factor of $1-\epsilon$.

b. Use dynamic programming to solve the knapsack problem in the modified instance. What is the resulting running time of your algorithm? Conclude that this results in an FPTAS for knapsack.

2 Bin Packing

Given n items with sizes $a_1, ..., a_n \in (0, 1)$, find a packing into bins of size 1 so that the total number of bins is minimized. Let B^* be the number of bins used in the optimal solution. As it is NP-hard to compute the optimal packing the goal is to develop an approximation algorithm.

a. Suppose that there are only $k = O(1)$ distinct item sizes for some constant k. Argue that you can solve bin-packing in polynomial time.
Hint: Use dynamic programming, keeping track of how many items of each size remain.

b. Suppose that you have packed all items of size greater than ϵ into B bins. Argue that in polynomial time you can add the remaining small items to achieve a packing using at most $\max\{B, 1 + (1 + 2\epsilon)B^*\}$ bins.

c. Consider grouping items by size. Fix some constant k, and let S_1 denote the largest n/k items, S_2 the next largest n/k, and so on. Suppose that you increase the size of each item to equal the largest size in its group, so that there are only k distinct sizes. Argue that this increases the optimal number of bins by at most n/k.
Hint: imagine setting aside the jobs in S_1. Argue that the remaining items, with their increased sizes, can still fit into the bins used by the original packing.

d. By applying this grouping procedure to items greater than $\epsilon/2$ for some $k = O(1/\epsilon^2)$, we can solve the resulting instance with k distinct item sizes optimally and then add the small items. Argue that this results in a polynomial time scheme that uses at most $1 + (1 + \epsilon)B^*$ bins.

3 Online Shopping

You want to buy a new laptop but want to send as little as possible for one that satisfies your needs. You plan to do your research but you realize that time is money. Every minute that passes costs you \$1. As you keep searching you find different websites with different prices. Let p_t be the price you will discover after searching for t minutes. The optimal strategy would be to stop at the time that minimizes $t + p_t$ and buy that laptop. Find a competitive strategy against that policy. You don't know the future prices but you assume that prices you found in the past will remain available for you to choose later on.

a. Show that the ski-rental problem is a special case of this problem.

b. Show that no online algorithm with competitive ratio lower than $e/(e-1)$ exists.
Hint: Use Yao's Minimax Principle for the ski-rental problem: Find a distribution of times where any deterministic stopping rule is at least $e/(e-1)$-competitive.

c. Consider the following strategy: At every time t where the optimum changes value, such that $t + p_t < t' + p_{t'}$ for $t' < t$, pick a random stopping time according to the ski-rental policy for buying cost $B = p_t$. Stop at that time unless the offline optimum changes again in which case you reset the stopping time accordingly. Show that this is $e/(e-1)$-competitive.

4 k-Server on Trees

In class we saw a k-competitive algorithm for k-server on the line. Show that the following extension is k-competitive in trees. For any request, we will show that a server is covered if there exists another server on its path to the request. To handle a request, start moving all uncovered servers towards the requested point at the same rate. Stop moving a server if it becomes covered or another server reaches the requested point.

a. Extend the analysis we did in class to show that this algorithm is k-competitive.

b. As we mentioned, paging can be seen as a k-server problem on the star graph. Show that any variant of the Marker algorithm that picks an arbitrary unmarked page to evict is k-competitive by drawing an analogy to the k-server algorithm above.